URL: http://www.elsevier.nl/locate/entcs/volume58.html 16 pages

Path-based Security for Mobile Agents

Gerald Knoll

Institute for Human & Machine Cognition, University of West Florida Pensacola, FL, USA

Niranjan Suri

Institute for Human & Machine Cognition, University of West Florida Pensacola, FL, USA

Jeffrey M. Bradshaw

Institute for Human & Machine Cognition, University of West Florida Pensacola, FL

1 Introduction

As mobile agents are increasingly adopted in intranets, on the Internet, and on computational grids, new security concerns become increasingly important. Unlike other kinds of mobile code, such as applets, which are pulled a single time to remote systems (single-hop), mobile agents may move using their own itinerary through a series of systems (multiple-hop), potentially carrying sensitive information with them. In such scenarios, mobile agents introduce new vulnerabilities: hosts are at the mercy of malicious (or buggy) agents that may compromise the integrity of the host execution environment and agents are at the mercy of malicious hosts that are capable of examining or modifying the code or data of the agent, or presenting a false environment to the agent (as in a masquerade attack), thereby causing the agent to execute incorrectly. Widely used security measures such as cryptography, authentication, and access control do not address these kinds of problems.

A particularly dangerous form of agent tampering by a malicious host occurs in the multiple-hop scenario. In this situation, an agent that was benign (and potentially trusted) to begin with could be turned into a malicious agent. Since the agent was originally a trusted agent, a subsequent host might naively grant the agent higher privileges, which could be misused by the now malicious agent. Such an agent could cause significantly more damage to the unsuspecting host than a malicious agent that was not trusted by the host and thereby executed with greater protections.

Some solutions have been developed to address both security concerns: protecting hosts from malicious agents and protecting agents from malicious

hosts. Note that it is also possible for a malicious agent to attack another agent. However, given that a malicious host has complete control over an agent's execution environment, the points of attack available to a malicious host comprise a superset of the points of attack available to a malicious agent. Therefore, the problem of protecting an agent from a malicious agent is subsumed by the problem of protecting the agent from a malicious host.

In general, the solutions to protect hosts are successful (if the hosts applies the necessary security mechanisms when executing agents). However, the solutions to protect agents have not been successful. The goal of this effort is not to solve the problem of protecting an agent from a malicious host, but to help a host determining the appropriate trust level of a visiting agent so that a host may protect itself using the necessary security mechanisms.

Section two of this paper provides a survey of existing solutions to mobile agent security. The following section briefly describes the NOMADS mobile agent environment where our proposed security solution is being implemented. Section four presents the proposed solution for path-based security for mobile agents. Section five concludes with a summary and a discussion of future work.

2 Mobile Agent Security

Agent security may be broadly divided into two categories: securing an agent and securing a host. Further, securing an agent may be classified into different threats and attacks. The main threats are tampering with the agent and extracting private information from an agent. Denial of service attacks against an agent are placed in the category of tampering with the agent. Masquerading attacks by a host or other agents are mentioned but not discussed in detail. Figure 1 summarizes the existing solutions for agent security.

2.1 Securing a Host

Several different solutions are available in the field of host security. Software-based fault isolation [24] (sandboxing [3]) and safe code interpretation [16] provide security mechanisms that successfully protect hosts against many different threats. Additionally, there are other solutions such as signed code [6], state appraisal [22], path histories [4,15], safe code interpretation [16], and proof carrying code [13,24] that provide partial host security. The NOMADS mobile agent environment is an example of a system that provides dynamic access and resource control and a policy-based approach to host security [2,20,21].

Host security mechanisms often have to make a tradeoff between the ability of an agent to perform operations and the security of the system. A highly restrictive environment (e.g. sandboxing in JDK 1.0.2) may provide good security but does not allow agents to carryout many useful tasks. On the other

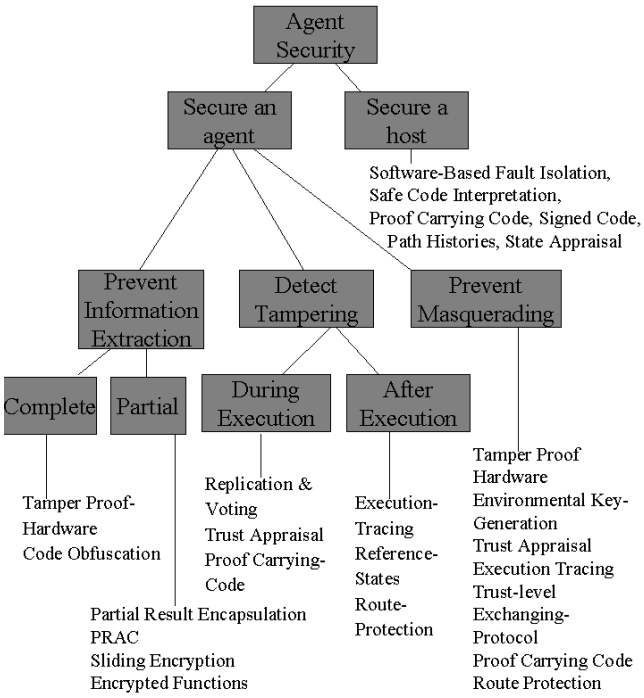

Fig. 1. Classification of Agent Security Approaches

hand, a less restrictive environment may allow agents to perform necessary tasks but may not be able to protect a host from a malicious agent. In an ideal world, by contrast, we would be able to determine exactly what level of trust is appropriate for a given agent, and determine the specific operations that should be allowed for that agent accordingly. NOMADS approaches this problem by allowing different security policies to be applied to an agent depending on various properties such as its owner or code base. However, in multiple-hop scenarios it is often difficult to determine an appropriate trust level for the agent. For example, consider an agent that enjoys a high level of trust and access to system resources because it has been created locally on the host and begins execution on behalf of the owner of that host. However, once the agent begins to visit other hosts, there is a possibility of agent tampering. When a tampered agent returns to the original host and the host reassigns a high trust level to the agent, then the modified agent could easily disrupt the integrity of the host's execution environment.

Given that there are no satisfactory solutions to detect tampering of an agent (see section 2.2), the agent could be manipulated by a malicious host without detection.

With respect to the protection of agents, there are several limited solutions, but ultimately no satisfactory ones. See Table 1 below for a summary.

Preventing information extraction. A goal of some approaches is to prevent any information from being extracted from the agent, thus making tampering difficult or at least detectable. Examples of such approaches include tamper proof hardware [26] and code obfuscation [8]. Tamper proof hardware requires a specially manufactured device that acts as the execution environment for an agent. The hardware device still can be modified after longer observations and therefore needs an additional maintenance service. Currently, there is no manufacturer who produces these devices for the masses. Also, the high cost of these devices makes them impractical for a large number of users. Even if tamper proof hardware devices were to become commonly available in the future, we would still need to ensure that the manufacturers do not have malicious intentions. Another approach is limited black box [8], which uses the concept of code obfuscation. In this case, performance decreases because of additional costs at transmission and execution time. No proofs have been developed to establish the minimum duration of time for which the methods are effective. Additional limitations are that a global time clock is needed and the fact that black box testing of the agent can still occur. The above approaches to change an agent into a black box provide no practical and completely implemented solution. The approach can be also be used to encrypt only parts of an agent in case the agents are large. Note that another unfortunate consequence of black box approaches is that the host, by virtue of not being able to examine the agent, might be less able to protect itself from a malicious agent.

Another category of approaches tries to secure a small part of the agent against information extraction and tampering by a host. Examples in this category include partial result encapsulation [9], sliding encryption [28], partial result authentication codes [27], forward integrity [1,10], environmental key generation [17], computing with encrypted functions [18], undetachable signatures [11], and trust appraisal [5]. Environmental key generation is useful if there is a need for different levels of access rights to an agent. If a given environmental condition is true, some encrypted data is decrypted for the host. However, a major problem with this approach is that it allows a host to explore the executable code of the visiting agent. Computing with encrypted functions is an approach that provides functions that can only be executed in the encrypted stateso far this approach only supports functions that accept polynomial or rational inputs. This means in practical terms that most parts of an agent cannot be encrypted in this fashion. Undetachable signatures is a variation of encrypted functions and uses the RSA scheme to protect a signature function. Unfortunately, it is not a workable solution for a multiple hop scenario. Trust appraisal requires negotiation between an agent and host through trust appraisal functions. However, the approach is not practical if an

agent travels to many hosts during the course of execution without returning to the home system regularly. Moreover, there exists no acceptable mechanism to distinguish between normal state execution and malicious state execution.

Detecting tampering. In contrast to mechanisms that prevent information extraction, other mechanisms have been developed to detect tampering (either during tampering or after the tampering has succeeded). Detection mechanisms aim at detecting illegal modification of code, state or execution flow of a mobile agent. The approaches available in this category are mutual itinerary recording [4], server replication [12], execution tracing [23], reference states [7], proof carrying code [13], route protection [25], and the trust-level exchanging protocol [14]. Itinerary recording is based on replicating an agent several times. The overhead of the itinerary recording makes this approach impractical. Moreover, as the size of an agent increases, the performance is also affected by the replication cost. The agent has to be duplicated and executed multiple times, where constant communication between the agents is necessary. The problem with server replication is that all agents are executed on the same platform and if the platform is malicious, the platform may tamper with more than one agent thereby causing this approach to fail. Also, the performance loss due to replication is higher than that of the previous solutions. There are several problems with execution tracing. The first problem is that the traces add substantial overhead. The second problem is that the owner must be suspicious enough to initiate a verification operation. The third problem is that the algorithm requires a trusted home to perform the re-computation of the trace. Finally, the approach does not detect invalid (or malicious) input that a host may provide to an agent. Reference states requires that each state of the computation of the agent be repeated twice. Moreover, the previous host can send the malicious input to the next host, which implies that the current host has to verify the input. Collaborating attacks of two hosts also cannot be detected by this approach. Proof carrying code is not implemented yet because of the following missing features: a standard formalism for establishing security policy, automated mechanisms for the generation of proofs, techniques for limiting the potentially large size of proofs, and a platform independent implementation. Trust-level exchanging protocol provides no help when the host and the TTP do not trust each other. We recognize that masquerading attacks by a host or other agents are another source of problems for which solutions have been proposed, but do not consider these approaches here.

2.3 Protecting the Path of an Agent

The security approach proposed in this paper relies on analysis of the path of a mobile agent. Several approaches have been developed to ensure the integrity of an agent's path. The properties or structure of the agent determine the particular approach to be employed. In all variations of these approaches, the goal is to prevent or detect modification of the planned or already tra-

versed path. For some agents the path can be hard- coded while for other agents the path cannot be determined ahead of time. It is easier to protect static path information than the information about a path that has yet to be determined. Various approaches make the simplifying assumption that the agent returns back home after traveling through different hops. But this is a specific assumption that does not hold for all agents. To provide protection of an agent's uncertain path, it is necessary to collect some information, such as the hostname or IP address, at each hop. This information can be encrypted to prevent extraction or tampering.

One solution that provides full security is to allow the agent to travel only to trusted hosts that are already registered with a certificate authority (CA). Secure routing [14] takes this approach and ensures that the travel path is restricted through routing policies. The agent is not allowed to travel to a host whose IP address is not registered with a CA. But this does not provide a solution for an agent that needs to travel to hosts that are not a priori certified.

An approach that requires the encryption of only small parts of the agent is called partial result encapsulation [9]. In general, there are three different ways to provide partial encryption. One way is for the agent to encrypt its data without help of another subject. The agent does not reveal the means by which the data was encrypted. Another is to rely on the encryption capabilities of an agent's host. The third and most popular solution to date is to use a Trusted Third Party (TTP) that provides a timestamp on a digital fingerprint of the results.

The Partial Result Authentication Codes (PRAC) [27] approach relies on three entities: the agent, the host, and the TTP. As the agent migrates to a host, there is an authentication protocol, which generates a symmetric key that the agent can use to encrypt information. The key has to be provided to the home or TTP in a secure way. After the execution of the agent, the symmetric key in the agent is destroyed so that the key is not misused. Back at home, the encrypted code can be analyzed to make sure that no host has changed anything and that forward integrity [1] is provided. Instead of using symmetric keys, it is also possible to use a Public Key Infrastructure (PKI) or digital signatures. A problem with this approach is that no backward integrity is provided. This means that the first host that is visited has access to the keys of all the hosts yet to be visited. When the agent visits the first host again, the host has access to all available data. There is also no mechanism to prevent the deletion of all of the encrypted data. A PRAC enhancement proposed in [10] is to construct a chain of encapsulated results from the previous host to the next. Every applied data is encrypted with the private key of the current host and a hash function is constructed that links to the applied data. Then, the public key of the home is additionally used to encrypt the data.

Route protection [25] is a solution that has the added advantage of providing anonymity for an agent. It requires a home system to compute an

encrypted message for every host that an agent is going to visit before the agent leaves the home system. This encrypted message contains the agent's path. The current host only has access to routing information for the home, the previous host, and the next host. Through digital signing of these nested messages, attacks that attempt to modify the route of the agent can be detected. Additionally, it is possible to extend the route, but with the restriction that the host extending the route must provide the signed routing information in a similar manner to the home system. The host that extended the route must again be visited after the route extension ends. Most, but not all, collaborating attacks can be detected through broadcasting mechanisms. One drawback is the cost of the calculation that must be performed before the agent is sent and once again be performed each time the route is extended. Also the routing information grows exponentially with the number of hosts visited. Finally, detecting malicious behavior of a host requires an additional TTP.

2.4 Summary of Existing Approaches to Agent Security

Table 1 summarizes existing solutions to secure an agent. All solutions to date contain an unacceptable number of weaknesses or limitations. Some do not have an available implementation. The entries in the table are further categorized according to the forms of attacks they try to prevent.

3 Overview of Nomads

The proposed security mechanisms are incorporated into the NOMADS mobile agent system. NOMADS supports strong mobility and safe agent execution [20,21]. Strong mobility allows the execution state of an agent to be captured and moved with the agent from one host to another. NOMADS uses the state-capture mechanism to go beyond strong mobility and also support forced mobility, which allows the system to move agents from one host to another by force (potentially in a completely transparent manner to the agent). Such forced mobility is extremely useful for situations that involve load balancing, process migration, systems shutting down, etc. Safe execution of agents is based on the ability of NOMADS to control the resources accessed and consumed by agents. The resource control mechanism allows control over the rate and quantity of resources used by agents. Dynamically adjustable limits can be placed on several parameters including the disk, network, and CPU. These resource control mechanisms complement Java's access control mechanisms and help in making the NOMADS system secure against malicious agents. NOMADS derives its unique capabilities from a custom Java Virtual Machine called Aroma [19].

The NOMADS system makes extensive use of security policies and can work in conjunction with high-level policy management tools and mechanisms,

Approach of Agent Security	Protect Information?	Detect Tampering?	Prevent Masquerading?	Implementation Available?
Secure Routing	Yes	Yes	No	Yes
Tamper Proof Hardware	Yes	Yes	Yes	Yes
Code Obfuscation	Yes (˜ time)	Yes (˜ time)	No	No
Partial Result Authentication Codes	Partial	Partial	No	Yes
Sliding Encryption	Partial	Partial	No	No
Chained Digital Signature Protocol	Partial	Partial	No	Yes
Chained MAC Protocol	Partial	Partial	Yes	Yes
Environmental Key Generation	Partial	Partial	Yes	No
Encrypted Functions	Partial	Partial	No	No
Undetachable Signatures	Partial	Partial	No	No
Trust appraisal	No	Yes	Yes	Partial
Mutual Itinerary Recording	No	Yes	No	Yes
Server Replication	No	Yes	No	Yes
Execution Tracing	No	Yes	Yes	No
Reference States	No	Yes	Yes ()	Yes
Proof Carrying Code	No	Yes	Yes ()	No
Route Protection	Partial	Partial	No	Yes
Trust-level exchanging protocol	No	Yes	Yes	No

Table 1
Summary of Existing Solutions to Secure against Malicious Hosts

such as those implemented in KAoS and the DARPA CoABS Grid [2]. Currently, policies are enforced on an agent based on the authentication of the agent (i.e., the owner of the instance of the agent) and the agent's code source (i.e., the author of the agent). Additional enforcement strategies are under development. The security policies are used to specify a wide range of constraints on the execution of the agent. NOMADS also supports dynamically

changing the policies that apply to an agent based on observed behavior of the agent.

4 Path-Based Security for Mobile Agents

The mechanisms described in this paper extend the security framework already available in NOMADS. Currently, NOMADS does not take into account the prior path of host visits of a mobile agent (thereby making the system vulnerable to agents that have been tampered by malicious hosts). The goal of this approach is to take advantage of the information regarding the path of a mobile agent in order to make NOMADS more secure in a multiple hop scenario. In particular, the path of an agent should be taken into consideration when applying security policies to mobile agents. This requires that the path of a mobile agent be determined. The mechanism developed for determining the path is a lightweight approach that requires very little infrastructure and provides good performance when compared to previously described approaches. In particular, the goal is to not completely protect an agent's path information from being modified, but to limit the damage that a tampered agent could cause to other hosts (i.e., the principle of acceptable losses).

One of the goals of the approach was to require minimal additional infrastructure. In particular, we did not want to rely upon the existence of a Certificate Authority (CA). Using a CA complicates the necessary infrastructure that needs to be established for mobile agents to be deployed. Moreover, if a CA is used, then there is additional overhead when an agent moves to a server that is in a different realm. In this situation, either the destination host has to simply assume that the CA is trusted (which is a security weakness) or has to go through a laborious process of establishing the trust relationship to the original CA. Therefore, not requiring a CA implies wider scalability of agents, which is a fundamental requirement for mobile agent systems. The path-based security approach consists of two components:

(i) The chained IP protocol that maintains the path information.

(ii) A policy manager that uses the path information to compute a trust level for the agent and apply an appropriate security policy.

As described in section two, current solutions to obtain the path information of an agent are either too restrictive, incomplete (not implemented), or too expensive (in terms of performance). Given the desire for a lightweight system with little additional infrastructure, the chained IP protocol is very simple and does not attempt to provide cryptographically secure path information. Instead, each host is allowed to append to the path information directly. However, the policy manager applies the notion of trust levels to determine the accuracy of the path information as recorded by the various hosts. Note that the ultimate goal of our approach is to compute an appropriate trust level for an agent, not to determine the actual path of the agent.

Hence, our lightweight approach is adequate. However, if more accurate path information is desired, the chained IP protocol can be combined with a TTP.

4.1 Chained IP Protocol

The responsibility of the chained IP protocol is to maintain information about the agent's path. The path information (hostname + IP address) is attached to the agent's metadata and carried with the agent. Each host that the agent visits adds a new record to the path information. This record contains the IP address of the host that has been visited. When an agent arrives at a host, the receiving host determines the source IP address of the connection from the network layer. The receiving platform then checks to make sure that the source host has correctly added its IP address to the path information. This is illustrated in Figure 2.

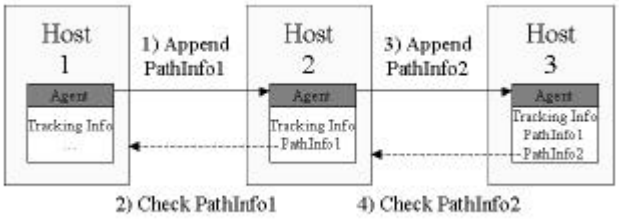

Fig. 2. Chained IP Protocol

In step 1, the path information of host 1 is added to the metadata (tracking information block) of the agent before the agent is sent to host 2. As the agent arrives at host 2, the host can check the IP address of host 1, to verify that the addresses matches with the path information in the agent. Steps 3-4 are the same as 1-2. Note that host 3 can only verify the IP address of host 2. In general, a receiving host can verify the validity of the path information only for the last hop. In particular, it does not prevent an intermediate host from changing previous path information (by adding spurious hosts to the path, masking or deleting hosts, or changing hosts). However, the implementation of the policy manager still allows an appropriate trust level to be computed regardless of such modifications (see section 4.2). Even multiple IP addresses on one host would only extend the configuration for the policy manager at every host.

This approach relies on the network layer's ability to correctly determine the IP address of the source of an agent transfer connection. Note that IP spoofing would allow a host to masquerade as another host thereby defeating the ability of the receiving host to correctly determine the source IP address. We are not assuming that IP spoofing will not occur. However, mechanisms to prevent IP spoofing or to make IP spoofing more difficult should be provided from the network administrator. IP verification is one possible way to

greatly reduce the possible scale of IP spoofing attacks. Figure 3 illustrates the reliability with which a receiving host can determine the previous path.

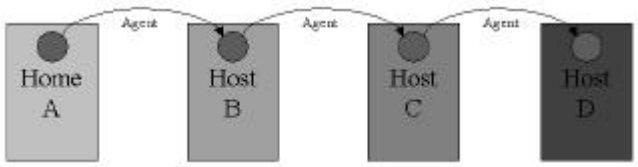

Fig. 3. Reliability of the Chained IP Protocol

Suppose an agent travels from host A to B to C before arriving at D. Host D knows that agent arrived from C (because of the information provided by the network layer). If the policy manager in D trusts host C, then the information verified by C (that the agent moved from B to C) can also be trusted. Similarly, if host B can be trusted, then the information regarding the agent's movement from A to B can also be trusted. Therefore, the reliability of the path information can be evaluated based on the trust levels of the preceding hosts. The reliability of the previous host information may become weaker as we get farther away from the current host.

If more reliable path information is desired, the chained IP protocol can be combined with a TTP. The operation of the TTP is illustrated in Figure 4.

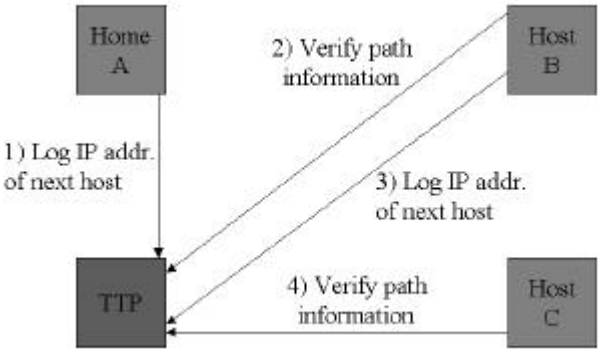

Fig. 4. Chained IP Protocol with a TTP

The TTP requires a few additional steps during the transfer of agents. Before an agent is sent to another host, the source host records the destination host's IP address with the TTP (as shown in steps 1 and 3). When a host receives an agent, the host may contact the TTP to verify the path information that is present in the agent (as shown in steps 2 and 4). Note that these steps are in addition to the steps described in Figure 2 to update the path information in the agent.

Figure 5 shows the reliability with which a receiving host can determine the previous path when a TTP is present. Note that if the receiving host does not trust the TTP, then we are back to the same case as above. However, if the receiving host trusts the TTP, then the path information recorded by the agent can be compared with the path information in the TTP. A TTP allows the algorithm to determine the path with a high degree of confidence even if one of the hosts is not trusted. It is still possible for two malicious hosts to collude with each other and falsify the path information between them. Therefore, the policy manager on the receiving host has to still check whether any of the intermediate hosts are not trusted and act accordingly.

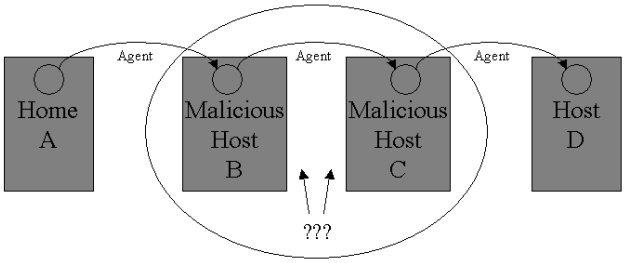

Fig. 5. Trust Levels with help of a TTP

An interesting possibility is that an intermediate host may lower the trust level of an agent by deliberately inserting IP addresses of untrusted hosts (although the agent may have never visited those hosts). However, such an action implies that the intermediate host is malicious and should not have been trusted by the policy manager anyway.

4.2 Policy Manager

The role of the policy manager is to compute the trust level of an agent and then choose an appropriate security policy. An administrator must provide the policy manager with information about the trust levels of hosts. This information is provided by means of a configuration file that maps IP addresses (or hostnames) to trust levels. By default, the policy manager assigns a low trust level to all hosts unless overridden by an entry in the configuration file.

The algorithm applied by the policy manager to compute the trust level is straightforward. The first check is to verify that the last hop address recorded in the path information is the same as that reported by the network layer. If this check fails, the policy manager generates a warning and returns with a low trust level.

Then, the policy manager traverses backwards through the path information and determines the trust level for each host. The final trust level is the minimum of all the trust levels of the hosts in the path. Figure 6 shows an example of an agent that moves from the home system (A) through hosts B and C before arriving at host D. On host B, host A is assigned a high trust

level. On host C, the trust levels are low and high for A and B respectively. On host D, the trust levels are medium, low, and high for A, B, and C respectively. Applying the above algorithm, host B will assign a high level of trust to the agent and hosts C and D will assign low levels of trust to the agent.

Note that the path information carried by the agent is not encrypted and is accessible to all hosts. This allows each host to independently compute a trust level based on the path and the local configuration information.

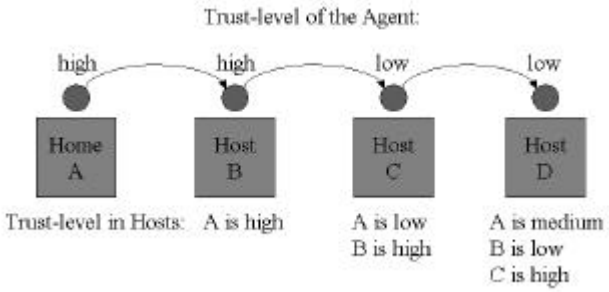

Fig. 6. Trust-level Calculation of an Agent

Figure 7 shows another example of the computation of the trust levels. As a new agent is created at Home A, the trust level of the agent is set to high (step 1). Home A sends the agent to Host B (step 2). Host B recomputes the trust level of the agent through its security policies. The trust level remains high (step 3) because B trusts all systems in the local network. Host B sends the agent into another environment, a famous bookstore, in this example (step 4). The bookstore does not know anything about the client network (and consequently host B). Therefore, the trust level is calculated as low (step 5). When host D receives the agent from host C (step 6), the trust level remains low (step 7). Host D sends the agent in step 8 back to the client network to home A. Because host A has a high trust level for the bookstore network, the trust level of the agent is recalculated to be high (step 9).

5 Conclusions and Future Work

The path-based security for mobile agents introduced in this paper provides a mechanism that extends the security of the NOMADS mobile agent system in a multiple-hops scenario. A lightweight protocol for tracking agent paths has been developed that is based on chaining IP addresses. A receiving host environment computes a trust level for the agent, which is then used to choose and apply a security policy to the incoming agent. A TTP is optionally supported to provide more reliable path information.

The current implementation uses just three levels of trust: high, medium, and low. In the future, we would like to extend the system to support a finer granularity of trust levels. Also, the current implementation automatically assigns a low level of trust to hosts that are unknown. We would like to explore

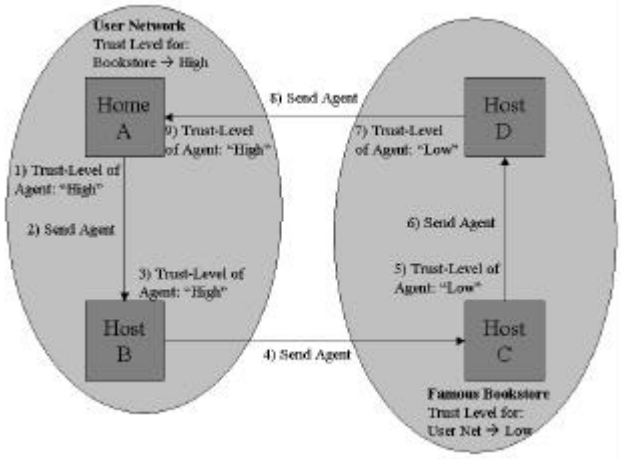

Fig. 7. Change of Trust Level

a mechanism that allows the trust levels to be derived through transitive trust relationships. Finally, we would like to incorporate mechanisms that dynamically vary the trust levels of hosts based on past historical information regarding the behavior of those hosts.

References

[1] Bellare, M., and Yee, B.S. Forward Integrity For Secure Audit Logs. Technical Report UC at San Diego, 1997.

[2] Bradshaw, J. M., Suri, N., Kahn, M., Sage, P., Weishar, D. and Jeffers, R. Terraforming cyberspace: Toward a policy-based grid infrastructure for secure, scalable, and robust execution of Java-based multi-agent systems. Proceedings of the Autonomous Agents 2001 Workshop on Scalable Agent Infrastructure, May 2001, Montreal Canada.

[3] Chang, F., Itzkovitz, A., and Karamcheti, V. Secure, User- level Resource-constrained Sandboxing. TR1999-795, 1999.

[4] Chess, D. Itinerant Agents for Mobile Computing. IEEE Personal Communications, vol. 2, no. 5, 1995, pages 34-49.

[5] Farmer, W., Guttman, J., and Swarup V. Security for Mobile Agents: Authentification and State Appraisal. Fourth European Symposium on Research in Computer Security, pages 118-130, 1996.

[6] Gray, R. S. Agent Tcl: A Flexible and Secure Mobile-Agent System. Proceedings of the 4th Annual Tcl/Tk Workshop, 1996, pages 9-23.

[7] Hohl, F. A Framework to Protect Mobile Agents by Using Reference States. Technical Report Nr. 2000/03, Universitt Stuttgart, 2000.

[8] Hohl, F. Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In Vigna, G. (Ed.), Mobile Agents and Security, Springer-Verlag, 1998.

[9] Jansen, W., and Karygiannis, T. Mobile Agent Security. NIST Technical Report, 1999.

[10] Karjoth, G., Asokan, N., and Guelcue, C. Protecting the computation results of Free-roaming agents. Rothermel, Popescu-Zeletin, pages 1-14, 1998.

[11] Kotzanikolaou, P., Burmester M., and Chrissikopoulos, V. Secure Transactions with Mobile Agents in Hostile Environments. In Dawson, E., Clark, A., and Boyd, C. Information Security and Privacy. LNCS Vol. 1841, Springer-Verlag, pages 289-297, 2000.

[12] Minsky, Y., Renesse, R. van, Schneider, F.B., and Stoller, S.D. Cryptographic Support for Fault-Tolerant Distributed Computing, 1996.

[13] Necula, G.C., and Lee, P. Safe, Untrusted Agents using Proof-Carrying Code. Carnegie Mellon University, 1998.

[14] Ng, S. Protecting Mobile Agents against Malicious Hosts. Master Thesis: The Chinese University of Hong Kong, 2000.

[15] Ordille, J. J. When Agents Roam, Who Can You Trust? Proceedings of the First Conference on Emerging Technologies and Applications in Communications, Portland, Oregon, May 1996.

[16] Ousterhout, J. K. Scripting: Higher-Level Programming for the 21st Century. IEEE Computer, 1998, pages 23-30.

[17] Riordan, J., and Schneier, B. Environmental Key Generation Towards Clueless Agents. Vigna, G. (Ed.), Mobile Agents and Security, Springer-Verlag, 1998.

[18] Sander, T., and Tschudin, C.F. Protecting Mobile Agents Against Malicious Hosts. Vigna G. (Ed.) Mobile Agents and Security. Springer-Verlag, 1997.

[19] Suri, N. Bradshaw, J.M., Breedy, M.R., Ford, K.M., Groth, P.T., Hill, G.A., Saavedra, R. State Capture and Resource Control for Java: The Design and Implementation of the Aroma Virtual Machine. Available on-line at http://nomads.coginst.uwf.edu/.

[20] Suri, N., Bradshaw, J.M., Breedy, M.R., Groth, P.T., Hill, G.A., Jeffers, R., and Mitrovich, T.S. An Overview of the NOMADS Mobile Agent System. Sixth ECOOP Workshop on Mobile Object Systems.

[21] Suri, N., Bradshaw, J.M., Breedy, M.R., Groth, P.T., Hill, G.A., and Jeffers, R. Strong Mobility and Fine-Grained Resource Control in NOMADS. Proceedings of the 2nd International Symposium on Agents Systems and Applications and the 4th International Symposium on Mobile Agents (ASA/MA 2000). Springer-Verlag.

[22] Swarup, V. Trust Appraisal and Secure Routing of Mobile Agents. DARPA Workshop on Foundations for Secure Mobile Code Workshop, 1997.

[23] Vigna, G. Cryptographic Traces for Mobile Agents. In Vigna, G. (Ed.): Mobile Agents and Security, pages 137-153, Springer-Verlag, 1998.

[24] Wahbe, R., Lucco, S., and Anderson, T. Efficient Software- Based Fault Isolation. Proceedings of the 14th ACM Symposium on Operating Systems Principles. ACM SIGOPS Operating Systems Review, 1993, pages 203-216.

[25] Westhoff, D., Schneider, M., Unger C., and Kaderali, F. Protecting a Mobile Agent's Route against Collusions. Springer LNCS 1758, 1999.

[26] Wilhelm, U.G., Staamann, S., and Butty, L. Introducing Trusted Third Parties to the Mobile Agent Paradigm. In Vitek J., and Jensen, C. (Eds.), Secure Internet Programming: Security Issues for Mobile and Distributed Objects, pp. 471-491, Springer-Verlag, 1999.

[27] Yee, B.S. A Sanctuary for Mobile Agents. DARPA Workshop on Foundations for Secure Mobile Code, 1997.

[28] Young, A., and Yung, M. Encryption Tools for Mobile Agents: Sliding Encryption in Biham, E. (Ed.): Fast Software Encryption. Springer-Verlag, 1997.